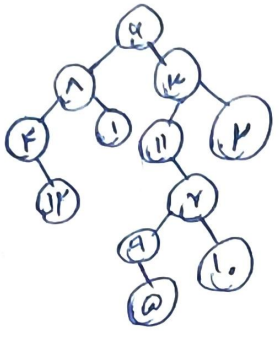
Postorder

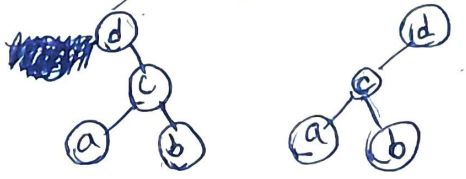
(الف)



ب) با داشتن preorder, Postorder هرگاه می‌توان یک درخت واحد کشید مثال نقض:

post: a b c d

pre: d c a b



ج) با inorder, preorder می‌توان درخت را کشید.

می‌دانیم اولین عنصر در preorder ریشه است پس در inorder عناصر سمت راست مربوط به زیردرخت راست و برعکس هستند حال وقتی زیردرخت راست و چپ پیدا شد مجدداً نزدیک‌ترین عنصر اولین عنصر را میان فرزند چپ یا راست ریشه محاط می‌کنیم مجدداً با پیدا کردن اینکه هر عنصر در زیردرخت چپ یا راست است (از روی inorder) مابقی عناصر را وارد درخت می‌کنیم.

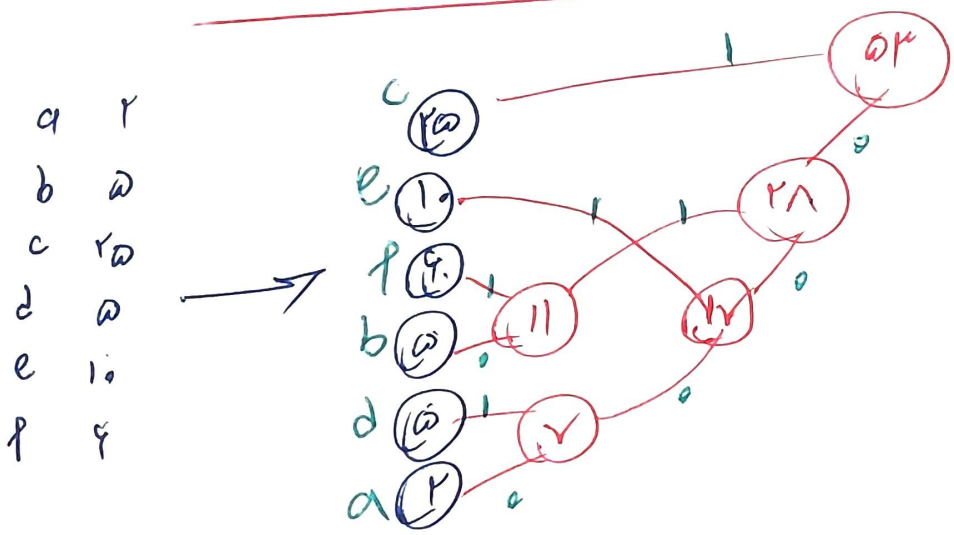
(۲) در ابتدا یک تک Node با مقدار برابر با پیشا ایجاد می‌کنیم O(1) (در نهایت یک لینکدیست با جمع ستون‌ها داریم)
در پیمایش پیش ترتیب اول به زیردخت چپ در نقطه اصلی رفتیم و مقدار Prev در لینکدیست جمع ستون‌ها Null بود
یک Node جدید ایجاد می‌کنیم و مقدار موجود در BST را در مقدار قبلی Node لینکدیست اضافه می‌کنیم
اگر در BST به سمت راست حرکت کردیم در لینکدیست یک خانه جلو (Next) می‌رویم در صورتی که
Null بود خانه‌ی جدید را ایجاد و مقدار دهی می‌کنیم (مقدار همان با مقدار موجود در خانه‌ی حال حاضر در BST)
اگر Node برای آن ستون وجود داشت نیز مقدار قبلی آن Node را با مقدار BST جمع می‌کنیم
در هر مرحله یک حفاظ برای بین مقدار نهایی Node لینکدیست و متغیر MAX که در ابتدا صفر است
انجام می‌شود اگر مقدار Node بیشتر از MAX بود، MAX با مقدار Node آپدیت می‌شود
چون تنها یکبار پیمایش پیش ترتیب انجام می‌شود الگوریتم از مرتبه‌ی O(n) است

---

(۳) ابتدا برای آرایه مقدار مجموع پیشوندی محاسبه می‌کنیم. به این ترتیب که هر خانه‌ی آرایه حاصل جمع تمامی خانه‌های ماقبل
آن و خود آن است. حال برای یافتن مقدار مجموع عناصر از a تا b (شرطی که
a<b باشد (a و b اندیس‌های آرایه هستند)) مقدار Arr[b] - Arr[a] را محاسبه می‌کنیم تا به مجموع
عناصر مابین a و b برسیم (عناصر برگرفته از a و کوچک‌تر مساوی b).
سپس با بیتی مجموع زیر آرایه‌های مختلف را وارد یک Min_heap کنیم. اگر تعداد عناصر داخل از m کمتر بود
عنصر جدید را در آرایه به داخل Push می‌کنیم. که از مرتبه O(log m) است
در غیر این صورت و وقتی که تعداد عناصر هرم به تعداد m رسید پس از خروج هر عنصر از آرایه اگر عنصر
از رئیس داخل بزرگ‌تر بود رئیس را Pop کرده و عنصر را Push می‌کنیم
و در غیر این صورت عدد از زیر آرایه بعدی می‌رویم (در هر مرحله یک عدد Arr[b]-Arr[a] : الگوریتم
داده می‌شود و این عدد به درخت Push می‌شود) چون برای یافتن همه‌ی Arr[b]-Arr[a] ها
دو حلقه‌ی for روی آرایه پیاده کردیم و در هر بار پیمایش O(log m) عمل Push می‌کنیم پس الگوریتم از O(n² log m)
است

(۴) دو عدد a و b به صورت داده می‌شود فرض می‌کنیم $b > a$ است. ابتدا دو متغیر تعریف می‌کنیم.
$max = b$ ، $min = a$ است در هر مرحله یکی از ۲ متغیر MAX ، min را به روزرسانی می‌کنیم.
از ابتدای آرایه شروع می‌کنیم و هر عنصر آرایه را با عنصر بعدی مقایسه می‌کنیم اگر مقدار عنصر مدنظر ما که روی آن هستیم از عنصر بعدی بیشتر بود مقدار MAX را به مقدار عنصر تغییر می‌دهیم (مثلا اگر مقدار عنصر از عنصر بعدی خود کمتر بود مقدار Min را به آن تغییر می‌دهیم. این عمل را روی عناصر آرایه یک بار پیمایش انجام می‌دهیم هرگاه در طول پیمایش مقدار MAX افزایش یافت یا مقدار min کاهش یافت بدین معنی است که رکورد جدیدی برای ایجاد شده پس دنباله‌ی داده شده قابل ایجاد در یک BST نیست!
همچنین چون روی عناصر آرایه یکبار پیمایش می‌کنیم الگوریتم از مرتبه‌ی $O(n)$ است

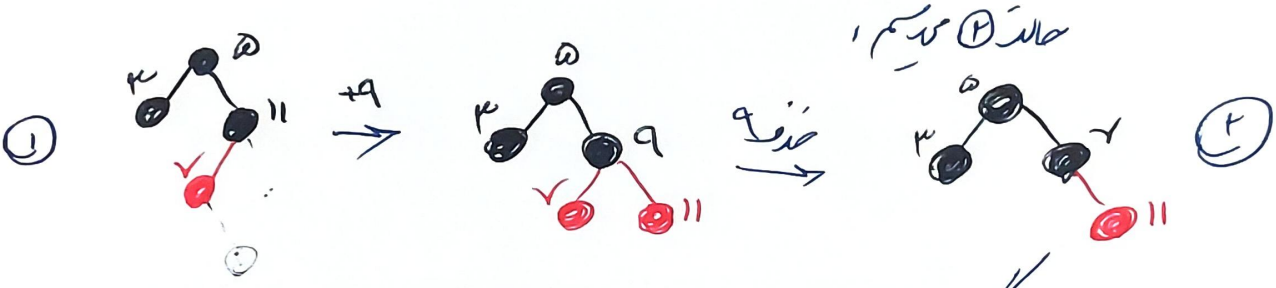
(۵)



a: 0000
b: 010
c: 1
d: 0001
e: 001
f: 011

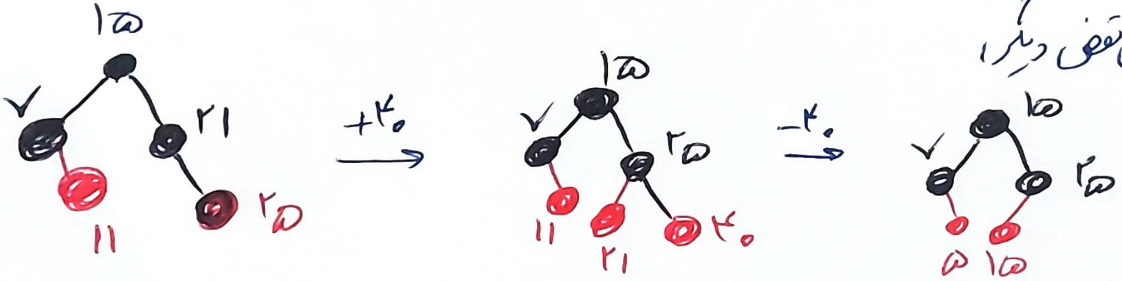
→ bdfce: 010001011100 1

ب) قسمت الف مثال نقض برای این موضوع است مثلاً برای اضافه کردن ۹ از حالت زیر شروع کنید.



حالت ② را رسم کنید،

مثال نقض دیگر،



ج) ابتدا ثابت می‌کنیم $2^{bh(x)}-1$ تعداد حداقل گره‌های داخلی هر زیردرخت دلخواه به ریشه x است!

استقراء، فرض ① اگر ارتفاع x صفر باشد، x برگ است $\Rightarrow 2^0-1=0$ ✓

گام ② فرض می‌کنیم برای همه‌ی xها با $bh(x)-1$ تا $bh(x)$ قضیه برقرار است!

برای هر گره‌ی x با توجه به فرزندانش اگر فرزندش سیاه باشد bh فرزند برابر $bh(x)-1$ است اما اگر قرمز باشد $bh(x)$ است. پس حداقل bh فرزندان $bh(x)-1$ است

طبق فرض استقراء برای هر کدام از ۲ فرزند تعداد گره‌های داخلی $(2^{bh(x)-1}-1)$ است چون ۲ فرزند و یک ریشه هم داریم پس تعداد گره‌های داخلی $1+(2^{bh(x)-1}-1)+(2^{bh(x)-1}-1)$ است پس تعداد گره‌های داخلی مساوی $2^{bh(x)}-1$ است!

می‌دانیم $bh(x) \geq \frac{h}{2}$ (که h ریشه باشد) چون می‌توان ۲ تا قرمز پشت سر هم را است پس h حداقل نصف کمتر سیاه هستند

$$\Rightarrow 2^{bh(x)}-1 \leq n \Rightarrow 2^{\frac{h}{2}}-1 \leq n \Rightarrow \frac{h}{2} \leq \log(n+1) \Rightarrow h \leq 2\log(n+1) \Rightarrow \boxed{h=O(\log n)}$$

(۴) درخت‌های AVL با ارتفاع h کمترین تعداد گره را وقتی خواهد داشت که یکی از زیردرخت‌ها درست ریشه ارتفاع $h-1$ و دیگری $h-2$ باشد

تعداد گره‌ها = تعداد گره زیردرخت چپ + تعداد گره زیردرخت راست + ۱

چون تعداد گره‌ها در زیردرخت با ارتفاع $h-2$ کمتر از زیردرخت با ارتفاع $h-1$ است!

پس:

$$2n(h-2) < n(h-1) + n(h-2) \Rightarrow$$

$$n(h) > 2n(h-2) > 4n(h-4) \rightsquigarrow n(h) > 2^i n(h-2i)$$

از حل رابطه‌ی بازگشتی داریم:

$$\boxed{n(h) > 2^{h/2}}$$

---

(۶) سوال ۲ از تمرین ۲:

```
Preorder( L:List, i:int){
    if i >= L.size:
        return;
    else if (L[i] = Null):
        return;
    print ( L[i]);
    preorder( L, 2i+1)
    preorder( L, 2i+2)
    }
```

```
postorder (L.List, i:int)
  if (i>L.size) or (i<0) or (L[i]=Null)
      return
  Postorder (L, 2i+1)
  Postorder L, 2i+2)
  Print ( L[i])
```

فرض آن است که فرزند چپ گره در $2i+1$ و فرزند راست آن در $2i+2$ است و درصورت عدم وجود فرزند Null داریم.